Artificial intelligence & Machine Learning

Project Documentation

1. Introduction

- **Project Title:** Rain Prediction Pipeline Flask
- **Team Members:**

- **A. Athreya** – Project Lead
- **D. Rohith Sai Anjan** – Machine Learning Developer
- **A. Tarun Sri Phani Pavan Kalyan** – Frontend Developer
- **D. Uday Venkat** – Backend Developer
- **Ch. Lavanya** – Tester / QA Engineer

2. Project Overview

Purpose:

The purpose of this project is to develop a web-based application that predicts whether it will rain based on weather parameters using Machine Learning techniques.

Features:

- Web-based Flask application
- Weather parameter input form
- Data preprocessing
- Machine learning prediction engine
- Result display (Rain / No Rain)
- Error handling
- Model performance evaluation

3. Architecture

• Frontend (User Interface Layer)

The frontend is implemented using HTML and Flask templates. It provides a simple web interface where users enter weather parameters such as temperature, humidity, pressure, and wind speed. The interface displays rainfall prediction results generated by the machine learning model. This layer focuses on user interaction and visualization of model outputs.

• Backend (Machine Learning Inference Layer)

The backend is developed using Python and Flask. It manages request handling, input validation, data preprocessing, and model inference. Trained machine learning models (stored as `.pkl` files) are loaded during application startup. User inputs are transformed using preprocessing pipelines and passed to the classifier to generate rainfall predictions.

• Data Layer

The data layer consists of historical weather data stored in CSV format used for model training. During deployment, serialized model artifacts and preprocessing objects are stored locally. No external database is used in the current version of the system.

The system follows a three-tier architecture:

User Interface → Flask Backend → Machine Learning Model

This architecture enables real-time rainfall prediction through a lightweight web application.

4. Setup Instructions

• Prerequisites

The following software and libraries are required to run the Rainfall Prediction Machine Learning system:

- Python 3.x
- pip (Python package manager)
- Jupyter Notebook (for model training and evaluation)
- Flask framework
- Required Python libraries:
    - NumPy
    - Pandas
    - Scikit-learn
    - Matplotlib
    - Seaborn
    - Joblib / Pickle

• Installation

Follow the steps below to set up the project locally:

1. Clone the GitHub repository:

```
git clone https://github.com/akondi-
athreya/Rain_Prediction_Pipeline_Flask.git
```

2. Navigate to the project directory:

```
cd Rain_Prediction_Pipeline_Flask
```

3. Install required dependencies:

```
pip install -r requirements.txt
```

4. (Optional) Run the Jupyter Notebook to train the model if model files are not available.
5. Start the Flask application:

```
python app.py
```

6. Open a browser and access:

```
http://127.0.0.1:5000
```

Environment Variables

No environment variables are required for this version. The application runs locally using pre-trained model files.

5.Folder Structure

• Client (User Interface Layer):

The client layer consists of HTML templates rendered using Flask. These templates provide the input form for weather parameters and display the rainfall prediction results. All UI files are organized inside the `templates` folder.

• Server (Machine Learning Application Layer):

The server layer is implemented in Python using Flask. It contains:

- `app.py` – Main application file handling requests and predictions
- Serialized ML model files (`.pkl`)
- Preprocessing objects (scaler, encoder, imputers)
- Logic for data preprocessing and model inference

This layer manages user requests, prepares input data, loads the trained machine learning model, and returns prediction results to the client.

6. Running the Application Locally

The Rainfall Prediction system runs as a single Flask application that serves both the user interface and the machine learning model.

To start the application locally:

1. Navigate to the project directory:

```
cd Rain_Prediction_Pipeline_Flask
```

2. Run the Flask server:

```
python app.py
```

3. Open a web browser and access:

```
http://127.0.0.1:5000
```

The Flask server handles:

- User interface rendering (HTML templates)
- Data preprocessing
- Machine learning inference
- Prediction result display

No separate frontend or backend servers are required.

7. API Documentation

The backend exposes a simple Flask-based interface for rainfall prediction.

Endpoint 1: Home Page

URL: /
Method: GET

Description:
Loads the web interface that allows users to enter weather parameters.

Request Parameters: None

Response:
Returns the HTML page containing the weather input form.

Endpoint 2: Rainfall Prediction

URL: /
Method: POST

Description:
Accepts weather input values, performs data preprocessing, and generates rainfall prediction using the trained machine learning model.

Request Parameters:

Parameter	Type	Description
Temperature	Float	Temperature value
Humidity	Float	Humidity level
Pressure	Float	Atmospheric pressure
Wind Speed	Float	Wind speed
Rainfall	Float	Previous rainfall (if applicable)

Example Request:

User submits weather values through the web form.

Example Response:

Prediction Result:

```
Rain Expected
```

or

```
No Rain Expected
```

Displayed directly on the web interface.

Error Handling:

- Displays error message for missing or invalid inputs.
- Prevents prediction if required parameters are not provided.

Notes:

This application uses form-based submission instead of REST APIs. Predictions are returned as rendered HTML pages.

8. Authentication

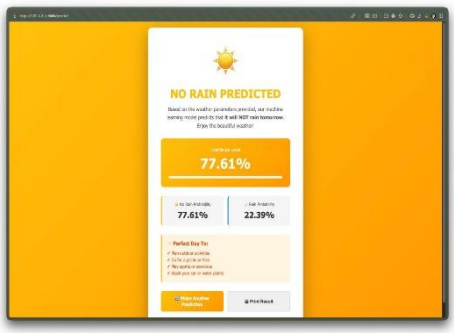
Authentication and authorization are not implemented in the current version of the Rainfall Prediction system.

The application is designed as an open web-based machine learning demo where users can directly access the prediction interface without login credentials.

No tokens, sessions, or role-based access controls are used. All users are allowed to input weather parameters and view prediction results.

This approach was chosen to keep the system simple and focused on demonstrating the machine learning workflow and prediction functionality.

9. User Interface



10. Testing

Testing Strategy and Tools Used

The Rainfall Prediction system was tested to ensure accurate model predictions, proper data processing, and smooth user interaction.

Testing Strategy:

1. **Data Validation Testing**
   Verified that user inputs are correctly validated before being passed to the model.
2. **Model Performance Testing**
   Evaluated the machine learning model using accuracy score, confusion matrix, and classification report to measure prediction quality.
3. **Functional Testing**
   Tested the Flask application to ensure correct handling of user requests, prediction generation, and result display.
4. **User Acceptance Testing (UAT)**
   Performed end-to-end testing to confirm the system works as expected from input submission to prediction output.
5. **Performance Testing**
   Checked application response time during prediction to ensure timely results.

Tools Used:

- Python
- Scikit-learn (model evaluation)
- Flask (application testing)
- Jupyter Notebook (model development and testing)
- Web browser (manual UI testing)

11. Screenshots or Demo

12. Known Issues

- The initial prediction may take a few seconds when the model loads for the first time.
- Model accuracy depends on the quality and size of the historical weather dataset.
- The system currently requires manual input and does not support real-time weather data.

- No user authentication is implemented in this version.
- The application is designed for academic use and may require optimization for large-scale deployment.

13. Future Enhancements

- Integrate real-time weather data using external APIs for live rainfall prediction.
- Improve model accuracy by training on larger and more diverse datasets.
- Display rainfall probability percentage along with Rain / No Rain output.
- Deploy the application on cloud platforms such as AWS or Heroku.
- Implement user authentication to store user prediction history.
- Add graphical visualization of weather trends and prediction results.
- Develop a mobile-friendly interface or Android application.
- Store prediction results in a database for further analysis.